实现平衡树

2019年8月21日 22:07

1. 结构体，下标实现treap

```
const int MN = 1e5 + 5;
const int inf = 0x7fffffff;

int n;
int tot=0,rt=1;     //结点数，根节点号
struct Treap{
     struct node{
          int l,r;      //左右子树结点下标
          int v,d;      //存储信息的数值，结点的平衡权值
          int cnt,sz;   //副本数、子树大小
     }t[MN];
     inline int ins(int x){     //插入值为x的新结点
          t[++tot].v =x;
          t[tot].d =rand();
          t[tot].cnt =t[tot].sz =1;
          t[tot].l =t[tot].r =0;
          return tot;
     }
     inline void up(int p){     //更新结点p的子树大小
          t[p].sz = t[ t[p].l ].sz + t[ t[p].r ].sz + t[p].cnt;
     }
     inline void init(){ //初始化根结点
          rt= ins(-inf);      //1
          t[rt].r = ins(inf); //2
          up(rt);
     }
     inline int getrk(int v,int p=rt){       //从值求下标
          if(!p) return 0;
          if(v== t[p].v) return t[ t[p].l ].sz +1;
          if(v< t[p].v) return getrk(v, t[p].l); //往左找
          return getrk(v, t[p].r) + t[ t[p].l ].sz + t[p].cnt;//往右找
     }
     inline int getv(int rk,int p=rt){       //从下标求值
          if(!p) return inf;
          if(t[ t[p].l ].sz >= rk) return getv(rk, t[p].l);    //往左找
          if(t[ t[p].l ].sz + t[p].cnt >= rk) return t[p].v;   //找到了
          return getv(rk - t[ t[p].l ].sz - t[p].cnt, t[p].r);//往右找
     }
     inline void zig(int &p){  //右旋，提起p的左儿子
          int q= t[p].l;
          t[p].l =t[q].r, t[q].r =p, p=q;
          up(t[p].r), up(p);  //更新原来的p(tp.r)和现在的p(原来的tp.l)
     }
     inline void zag(int &p){  //左旋，提起p的右儿子
          int q= t[p].r;
          t[p].r =t[q].l, t[q].l =p, p=q;
          up(t[p].l), up(p);  //更新原来的p(tp.l)和现在的p(原来的tp.r)
     }
```

```
inline void insert(int v, int &p=rt){  //沿着p插入v
     if(!p) p= ins(v);
     else if(v== t[p].v) ++t[p].cnt;
     else if(v< t[p].v){
          insert(v, t[p].l);
          if(t[p].d < t[ t[p].l ].d) zig(p);
     }else{ //v> t[p].v
          insert(v, t[p].r);
          if(t[p].d < t[ t[p].r ].d) zag(p);
     }
     up(p);
}
int pre(int v){     //返回值v的前驱值
     int ansp =1; //ans的节点号，初值负无穷结点
     int p =rt;   //用p遍历树，初值也是负无穷结点
     while(p){
          if(v== t[p].v){     //找到了v对应结点
               if(t[p].l){
                    p= t[p].l;   //沿左子树找
                    while(t[p].r) p= t[p].r;  //找最右的
                    ansp =p;
               }
               break;
          }
          if(t[p].v <v && t[p].v > t[ansp].v) ansp =p;
          p= v<t[p].v? t[p].l: t[p].r;
     }
     return t[ansp].v;
}
int nxt(int v){     //返回值v的后继值
     int ansp =2; //ans的节点号，初值正无穷结点
     int p =rt;   //用p遍历树，初值暂时是负无穷结点
     while(p){
          if(v== t[p].v){     //找到了v对应结点
               if(t[p].r){
                    p= t[p].r;   //沿右子树找
                    while(t[p].l) p= t[p].l;  //找最左的
                    ansp =p;
               }
               break;
          }
          if(t[p].v >v && t[p].v < t[ansp].v) ansp =p;
          p= v<t[p].v? t[p].l: t[p].r;
     }
     return t[ansp].v;
}
void del(int v,int &p=rt){//沿着p移除一个v值对应结点
     if(!p) return;      //
     if(v== t[p].v){     //找到了v对应结点
          if(t[p].cnt >1)     //只需减少一个副本
               --t[p].cnt, up(p);
          else if(t[p].l || t[p].r){//v值非叶结点
               if(!t[p].r || t[ t[p].l ].d > t[ t[p].r ].d)
                    zig(p), del(v, t[p].r);   //先提起左儿子，再删p
               else
                    zag(p), del(v, t[p].l);   //先提起右儿子，再删p
```

```
                up(p); //更新刚被提起来的儿子
            }else //删叶结点
                p =0;
            return ;
        }//else 没找到
        v<t[p].v ? del(v, t[p].l): del(v, t[p].r);
        up(p);
    }
}tr;

int main(int argc, char** argv) {
    tr.init();          //记得建根！！
    int n; scanf("%d",&n);
    while(n--){
        int op,x;
        scanf("%d%d",&op,&x);
        switch(op){
            case 1:     //插
                tr.insert(x);       break;
            case 2:     //删
                tr.del(x);          break;
            case 3:     //数值=>下标
                printf("%d\n",tr.getrk(x)-1);    break;
            case 4:     //下标=>数值
                printf("%d\n",tr.getv(x+1));     break;
            case 5:     //前驱
                printf("%d\n",tr.pre(x)); break;
            case 6:     //后继
                printf("%d\n",tr.nxt(x)); break;
        }
    }
    return 0;
}
```

2. STL::vector乱搞

```
vector<int>v;
inline auto l(int x){ return lower_bound(v.begin(),v.end(),x); }
inline auto u(int x){ return upper_bound(v.begin(),v.end(),x); }

int main(int argc, char** argv) {
    int n; scanf("%d",&n);
    while(n--){
        int op,x;
        scanf("%d%d",&op,&x);
        switch(op){
            case 1:     //插
                v.insert(l(x), x);  break;
            case 2:     //删
                v.erase(l(x));              break;
            case 3:     //数值=>下标
                printf("%d\n",l(x)-v.begin()+1); break;
            case 4:     //下标=>数值
                printf("%d\n",v[x-1]);          break;
            case 5:     //前驱
                printf("%d\n",*(--l(x))); break;
            case 6:     //后继
                printf("%d\n",*u(x));           break;
        }
```

```
    }
    return 0;
}
```